1980

Some Computational Aspects of Queueing Network Models

Herb Schwetman

Report Number:
80-354

Schwetman, Herb, "Some Computational Aspects of Queueing Network Models" (1980). *Department of Computer Science Technical Reports*. Paper 285.
<https://docs.lib.purdue.edu/cstech/285>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries.
Please contact epubs@purdue.edu for additional information.

Some Computational Aspects of Queueing Network Models

By

Herb Schwetman

Department of Computer Science

Purdue University

West Lafayette, IN 47907

CSD-TR 354

December 1980

February 1981 (Rev)

Note: This paper will appear in the Proceedings of the ORSA-TIMS Special Meeting on Applied Probability-Computer Science, the Interface, Jan. 5-7, 1981, at Florida Atlantic University, Boca Raton, FL.

Some Computational Aspects of Queueing Network Models

Herb Schwetman

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

ABSTRACT

Queueing network models have emerged as useful models of computer systems. While such models have been known and studied for some time in the area of operations research, their application to the study of computers has stimulated further research, both into extending the basic results and into developing efficient techniques for their solution. Currently, there does not appear to be any "best" technique. This paper surveys three well-known methods for solving queueing network models and discusses their complexity, size and numerical properties.

Introduction

Queueing network models (QNM's) have been studied and used for some time in the area of operations research. Since around 1968, these models have been used as models of computer and communications systems. This new use has stimulated research into enlarging the class of models which can be solved and into developing new solution techniques [KiSe79]. These new techniques

demonstrate improvements with respect to either complexity, size (when implemented in computer programs) and/or numerical properties.

Because of the complexity of networks of interest and of the solution algorithms, these techniques are often implemented as computer programs. In fact, several packages have been developed to solve fairly general classes of QNM's [Buze78]. Each of these packages has an network definition language, a solution portion, implementing some solution algorithm, and produces output, in the form of tables. The topic, computational aspects, is cast in the framework of these solution programs.

Developers and users of QNM solution programs are concerned with several issues, including:

- What are the types of models solved?
- Is the program giving correct answers?
- Is the program susceptible to problem-dependent numerical errors?
- What are the time and space requirements for the program?

This paper begins by introducing QNM's and the associated notation. Their use as models of computer systems is illustrated with an example. Three solution techniques, all developed since 1970, are described. This paper discusses each technique, with respect to these issues. It will be seen that there is currently no technique which is "best" with respect to all desirable attributes.

QNM's: Notation and an Example

In its simplest form, a QNM consists of a collection of K devices (also called facilities); jobs (tasks or customers) travel from one device to another, selecting the next device to visit according to the transition probabilities, q_{ij} , where q_{ij} is the probability that a job at device i will next visit device j . The subscript 0 denotes arrivals from or departures to "outside of the system". When jobs arrive

at a device, they wait until the single server is idle; waiting jobs are selected for service using a first-come, first-served (FCFS) scheduling discipline. The time spent in service at device i is governed by a negative exponential distribution with mean S_i . Queueing networks can be either open or closed. In a closed model, there are N jobs circulating amongst the K devices (i.e. no external arrivals). In an open model, jobs arrive at the network at rate X_0 . Since equilibrium must be maintained, these jobs also depart at the same rate.

A QNM of this form can be solved, to obtain values for the following performance variables: for device i

X_i - throughput rate (jobs per unit time),

U_i - utilization,

$\bar{n}_i$ - mean queue length, and

R_i - mean response time.

Figure 1

Example of Single Class, Closed QNM [Buze73]

These QNM's can be solved even when the descriptions of the models are enlarged to include:

- (1) a total of four different scheduling disciplines at the facilities : FCFS - first-come, first-served, IS - infinite servers (no delay), PS - processor sharing, and LCFSPR - last-come, first-served preempt;
- (2) "general" service time distributions (distributions with rational Laplace transforms) for stations which are not FCFS;
- (3) multiple classes of jobs, where a job class is described in terms of its transition probabilities, mean service intervals, population and whether it is open or closed;
- (4) for facilities which are not FCFS, jobs in different classes can have different mean service times;
- (5) class switching - jobs can switch between classes, subject to constraints involving flow conservation (e.g. jobs which are external arrivals must eventually depart the network); and
- (6) the service time at a device can depend on the number of jobs at that device (in particular devices with multiple servers can be modeled).

Thus, the most general form of QNM's for which efficient solution techniques are known to exist can be described as follows:

K - number of facilities, usually indexed by i or j ,

R - number of job classes, usually indexed by r or s ,

$q_{i,j,s}$ - probability that a job of class r at device i joins class s at device j ,

$S_{ir}(n_i)$ - mean service time at device i for jobs of class r , when n_i jobs are present, and

$N = (N_1, \dots, N_R)$ - a population vector with N_r jobs of class r (closed) in the model.

For each open class s , an arrival rate, X_{0s} must be specified.

For models of this more general form, a solution consists of device statistics by job class; for jobs of class τ , the results at device i are:

$X_{i\tau}$ - class throughput rate,

$U_{i\tau}$ - class utilization,

$\bar{n}_{i\tau}$ - class mean queue length, and

$R_{i\tau}$ - class mean response time.

Given these, we can then calculate device statistics, as follows:

$$U_i = \sum_{\tau} U_{i\tau}$$

$$X_i = \sum_{\tau} X_{i\tau}$$

$$\bar{n}_i = \sum_{\tau} \bar{n}_{i\tau}$$

$$R_i = \frac{\bar{n}_i}{X_i}$$

Solution Techniques

While QNM's with multiple classes can be solved, the notation becomes very cumbersome. Thus, except as noted, the remainder of this paper will consider only closed, single class QNM's. The landmark paper by Baskett, Chandy, Muntz and Palacios [BCMP75] gives the closed form solution for the general class of QNM's. The paper by Bruell, Balbo and Schwetman [BBS77], among others, gives details about solving multiple class models.

Closed, single class QNM's are completely characterized by the following parameters:

K = number of devices,

N = number of jobs,

$S_i(n)$ = the reciprocal of the service rate of device i , when there are n jobs present, and

either

V_i = the number of visits to device i per job,

or

q_{ij} = the probability that a job at device i next visits device j .

In this discussion, an index of 0 denotes "the outside world"; this is intended to represent the arrival of jobs from outside of the system and the departure of completed jobs. For example, V_0 is used to "normalize" the visits per job to the devices of the system. It can be shown that the $\{V_i\}$ can be obtained as the solution to the set of equations:

$$V_0 = 1$$

$$V_j = \sum_{i=0}^K V_i q_{ij} \quad \text{for } j = 1, \dots, K,$$

The V_i are unique up to a multiplicative constant. If the model is open, then X_0 is the arrival rate to the model; in this case,

$$X_j = \sum_{i=0}^K X_i q_{ij}$$

are uniquely determined, and

$$V_i = \frac{X_i}{X_0}.$$

Let

$$\underline{n} = (n_1, \dots, n_K)$$

be a state of a model, where n_i is the number of jobs at device i . Then

$$S(N, K) = \{\underline{n} = (n_1, \dots, n_K) | n_i \geq 0 \text{ and } \sum_{i=1}^K n_i = N\}$$

is the set of feasible states for a closed QNM. The probability of finding the model in state $\mathbf{z}$ is

$$p(\mathbf{z}) = \frac{1}{G(N)} \prod_{i=1}^K f_i(n_i)$$

where

$$f_i(n) = \begin{cases} 1 & n = 0 \\ V_i S_i(n) f_i(n-1) & n > 0 \end{cases}$$

and $G(N)$ is a normalizing constant. For devices with IS scheduling, $f_i(n)$ can be given as

$$f_i(n) = \frac{(V_i S_i)^n}{n!}$$

Because the $p(\mathbf{z})$ must sum to 1.0, the $G(N)$ can be calculated as

$$G(N) = \sum_{\mathbf{z} \in S(N,K)} \prod_{i=1}^K f_i(n_i)$$

While this is a solution, the number of terms in the sum render it impractical. Denning and Buzen [DeBu78] (among others) give the number of feasible states as

$$\binom{N+K-1}{K-1}$$

If $K = 8$ and $N = 20$, then the calculation of $G(N)$ requires the summation of 888030 terms, each the product of 8 factors.

Convolution Algorithm

Beginning with Buzen [Buze73], several researchers [ReKo75] [BaBS77] have shown that $G(N)$ can be computed without recomputing each factor of the sum. To do this, let

$$G_k(n) = \sum_{\mathbf{z} \in S(n,k)} \prod_{i=1}^k f_i(n_i).$$

Then $G_K(N) = G(N)$. We can factor $G_k(n)$ as follows:

$$\begin{aligned} G_k(n) &= \sum_{j=0}^n \left[\sum_{n \in S(n,k) \text{ and } n_k=j} \prod_{i=1}^k f_i(n_i) \right] \\ &= \sum_{j=0}^n f_k(j) \left[\sum_{n \in S(n-j,k-1)} \prod_{i=1}^{k-1} f_i(n_i) \right] \\ &= \sum_{j=0}^n f_k(j) G_{k-1}(n-j) \end{aligned}$$

If we define the $N+1$ dimensional vectors

$$f_k = (f_k(0), \dots, f_k(N))$$

and

$$G_0 = (1, 0, \dots, 0)$$

then the vector

$$G_k = (G_k(0), \dots, G_k(N))$$

with $G_k(n)$ as defined above is the vector convolution

$$G_k = f_k * G_{k-1}$$

If a device k has a single load independent server, then the elements $G_k(n)$ can be computed as

$$G_k(n) = V_k S_k G_k(n-1) + G_{k-1}(n).$$

This is Buzen's algorithm as given in [Buze73]. Bruell and Balbo [BrBa80] give a very complete treatment of the convolution algorithm. In addition to the solution for the single class model given here, they also present the convolution algorithm for multiple classes.

The required performance values can be calculated using the normalizing constants and auxiliary functions; the throughput rate at device i for load N is

$$X_i(N) = V_i \frac{G(N-1)}{G(N)}$$

For devices with load independent servers, the utilization is

$$U_i(N) = S_i X_i(N)$$

and the mean queue length is

$$\bar{\pi}_i(N) = \sum_{j=1}^N (V_i S_i)^j \frac{G(N-j)}{G(N)} = \frac{1}{G(N)} \sum_{j=1}^N (V_i S_i)^j G(N-j)$$

For devices with load dependent servers, define $P_i(n, N)$, the probability of having n jobs at device i when the load is N as

$$P_i(n, N) = \frac{f_i(n)}{G(N)} g_K^i(N-n)$$

where

$$g_K^i(n) = \begin{cases} G_K(0) = 1.0 & n = 0 \\ G(N) - \sum_{j=1}^n f_i(j) g_K^i(N-j) & n > 0 \end{cases}$$

Then, for these load dependent devices,

$$U_i(N) = \begin{cases} 1 - \frac{g_K^i(N)}{G(N)} & i < K \\ 1 - \frac{G_{K-1}(N)}{G(N)} & i = K \end{cases}$$

and

$$\bar{\pi}_i(N) = \begin{cases} \sum_{j=1}^N j P_i(j, N) & i < K \\ \frac{1}{G(N)} \sum_{j=1}^N j f_K(j) G_{K-1}(N-j) & i = K \end{cases}$$

For all devices, the mean response time is

$$R_i(N) = \frac{\bar{\pi}_i(N)}{X_i(N)}$$

If device K is a load dependent device (i.e. the last device), then the auxiliary functions $g_K^i(n)$ and $P_i(n, N)$ are not required for that device, as the alternative forms of the equations for $U_i(N)$ and $\bar{\pi}_i(N)$ can be used. Buzen [Buze78] used

this feature to develop a variation of the convolution algorithm which does not require the auxiliary functions; rather, the model is repetitively solved with each load dependent device moved to the K -th position. This procedure, called "reindexing", while requiring more computational effort, has better numerical properties than the standard convolution technique.

Mean Value Analysis

Another technique has been developed by Reiser and Lavenberg [ReLa78]. This technique, called Mean Value Analysis (MVA) does away with the need for calculating the normalization constant. The critical formulae, for load independent devices, are [BuDe80]:

$$R_i(n) = \begin{cases} S_i [1 + \bar{\pi}_i(n-1)] & \text{if FCFS, PS, or LCFSPR} \\ S_i & \text{if IS} \end{cases}$$

Note, the $\bar{\pi}_i(0)$ are initialized to 0.

The overall throughput rate then is

$$X_0(n) = \frac{n}{\sum_{j=1}^K V_j R_j(n)}$$

and the device throughput rate and mean queue length are

$$X_i(n) = X_0(n) V_i$$

$$\bar{\pi}_i(n) = X_i(n) R_i(n) .$$

For devices which are load dependent, the equation for the mean response time is modified to be

$$R_i(n) = \sum_{l=1}^n l S_i(l) P_i(l-1, n-1) ,$$

and $P_i(l, n)$ is given by

$$P_i(l, n) = X_i(n) S_i(l) P_i(l-1, n-1) \quad \text{for } l=1, \dots, n,$$

and

$$P_i(0, n) = 1 - \sum_{l=1}^n P_i(l, n)$$

These formulae lead to an algorithm which builds up the values of the desired performance variables from $n = 0$ to $n = N$. For load independent devices, the utilization is given by

$$U_i(N) = X_i(N) S_i,$$

and for load dependent devices,

$$U_i(N) = 1 - P_i(0, N)$$

As presented in [ReLa80], the MVA algorithm has a slightly different form, which can result in an increase in the storage and a decrease in the number of operations required. Briefly, this modified MVA algorithm is as follows:

$$W_i(n) = V_i R_i(n) = \begin{cases} V_i S_i [1 + \bar{\pi}_i(n-1)] & \text{if FCFS, PS, or LCFSPR} \\ V_i S_i & \text{if IS} \end{cases}$$

$$X_0(n) = \frac{n}{\sum_i V_i R_i(n)} = \frac{n}{\sum_i W_i(n)}$$

$$\bar{\pi}_i(n) = X_i(n) R_i(n) = X_0(n) V_i R_i(n) = X_0(n) W_i(n)$$

For load dependent devices,

$$W_i(n) = V_i R_i(n) = \sum_{l=0}^n l V_i S_i(l) P_i(l-1, n-1)$$

Also, Resier and Lavenberg present a special form of the load dependent device equations for multiple server devices which reduces the number of factors in these equations.

The MVA algorithm can be extended to handle models with multiple job classes. Also, both open and closed classes can be accommodated. A technical

report by Schwetman [Schw80b] describes computational procedures for all of these extensions, plus efficient procedures for managing storage for these.

LBANC Algorithm

Chandy and Sauer have developed a new algorithm [ChSa80], called Local Balance Algorithm for Normalizing Constants (LBANC). LBANC is similar to the convolution algorithm in that it produces normalizing constants, $G(n)$; it differs by using a technique similar to MVA to do this. Let $\bar{m}_i(n)$ be the unnormalized mean queue length at device i , i.e.

$$\bar{m}_i(n) = \bar{\pi}_i(n) G(n)$$

Since

$$\sum_{i=1}^K \bar{\pi}_i(n) = n,$$

we have

$$G(n) = \frac{\sum_{i=1}^K \bar{m}_i(n)}{n}.$$

To obtain the $\bar{m}_i(n)$, apply Little's Law to the MVA equation for $R_i(n)$, to get for load independent devices with FCFS, PS or LCFSPR scheduling

$$\bar{\pi}_i(n) = X_i(n) R_i(n) = X_i(n) S_i [1 + \bar{\pi}_i(n-1)],$$

or

$$\begin{aligned} \bar{m}_i(n) &= G_i(n) X_i(n) S_i [1 + \bar{\pi}_i(n-1)] \\ &= V_i S_i [G(n-1) + \bar{m}_i(n-1)]. \end{aligned}$$

The last equation follows from the definition of $X_i(n)$ from the convolution algorithm. For devices with IS scheduling,

$$\bar{m}_i(n) = V_i S_i G(n-1).$$

By initializing

$$G(0) = 1$$

and

$$\bar{m}_i(0) = 0 \quad i = 1, \dots, K$$

and then successively computing the $\bar{m}_i(n)$ and $G(n)$ for $n = 1, \dots, N$. $\bar{m}_i(N)$ and $G(N)$ are obtained. The performance variables are then calculated as

$$\bar{n}_i(N) = \frac{\bar{m}_i(N)}{G(N)}$$

$$X_i(N) = V_i \frac{G(N-1)}{G(N)}$$

$$U_i(N) = X_i(N) S_i$$

$$R_i(N) = \frac{\bar{n}_i(N)}{X_i(N)}$$

For devices with load dependent servers, the calculation of $\bar{m}_i(n)$ is modified as follows:

$$\bar{m}_i(n) = \sum_{l=1}^n l p_i(l, n)$$

where $p_i(l, n)$, the unnormalized queue length probability, is given by

$$p_i(l, n) = V_i S_i(n) p_i(l-1, n-1)$$

and

$$p_i(0, n) = G(n) - \sum_{l=1}^n p_i(l, n) .$$

For these devices, the utilization is given by

$$U_i(N) = 1 - \frac{p_i(0, N)}{G(N)} .$$

Computational Aspects

As these algorithms are implemented as programs, several issues arise. One is the amount of resources (the amount of storage and the amount of computational effort) required to solve QNM's. Thus, it is of interest to evaluate each algorithm in these terms. In order to do this, it is necessary to state our chosen measures of resource usage. In this paper, these will be the size of arrays and the number of arithmetic operations required. The evaluation is restricted to single-class, closed models with N jobs and K devices; α of these devices have load independent servers, and β have load dependent servers ($\alpha + \beta = K$).

It is assumed that the implementation of each algorithm will require the same input values; an estimate of the space required to store these is given in Table 1. Also assume that each solution will consist of $U_i(N)$, $X_i(N)$, $\bar{\pi}_i(N)$ and $R_i(N)$ for each of the K devices; these will be computed but not stored.

item	storage
V_i	$\alpha + \beta$
S_i	α
$S_i(N)$	βN
Load indep.	2α
Load dep.	$\beta (1 + N)$

Table 1

Storage for Inputs

The amount of storage and the number of operations for the convolution algorithm are estimated in Table 2. Tables 3 and 4 present the estimated storage requirements and operation counts for the MVA and the modified MVA algorithms. Table 5 gives the estimated storage and number of operations for the LBANC Algorithm.

In Tables 2, 3, 4 and 5, the total amount of storage and the estimated total number of operations are obtained by summing the load independent and load dependent estimates. Table 6 gives an example of a model, along with the

item	storage	no. operations
$V_i S_i$	α	α
$f_i(N)$	$\beta(N+1)$	$2\beta N$
$G(N)$	$N+1$	$2\alpha N + 2\beta(N^2 + N)$
$g_k^i(N)$	$\beta(N+1)$	$2\beta N$
$X_i(N)$		$\alpha + \beta$
$U_i(N)$		$\alpha + 2\beta$
$\bar{\pi}_i(N)$		$3\alpha N + 3\beta N$
$R_i(N)$		$\alpha + \beta$
Load indep.	$\alpha + N + 1$	$\alpha(5N + 4)$
Load dep.	$2\beta(N+1)$	$\beta(2N^2 + 9N + 1)$

Table 2

Convolution Method - Storage and Operations

storage and operation counts for the three algorithms.

item	storage	no. operations
$R_i(N)$	$\alpha + \beta$	$2\alpha N + \frac{3\beta}{2}(N^2 + N)$
$P_i(l, N)$	βN	$2\beta(N^2 + N)$
$P_i(0, N)$	β	$\frac{\beta}{2}(N^2 + N)$
$X_0(N)$		$2\alpha N + 2\beta N$
$X_i(N)$	$\alpha + \beta$	$\alpha N + \beta N$
$\bar{\pi}_i(N)$	$\alpha + \beta$	$\alpha N + \beta N$
$U_i(N)$		$\alpha + \beta$
Load indep.	3α	$2\alpha(3N + 1)$
Load dep.	$\beta(N + 4)$	$2\beta(2N^2 + 4N + 1)$

Table 3

MVA Method - Storage and Operations

It can be noted that all of the presented algorithms require only modest amounts of storage. However, these storage requirements expand rapidly as multiple job classes are introduced. A similar comment can also be made about the execution times.

Also to be considered is the range of applicability of the different algorithms. Chandy and Sauer [ChSa80] discuss the generality of these three algorithms, stating that "convolution is the only algorithm which has been applied to the full class of product form networks". Schwetman [Schw80b] has shown how

item	storage	no. operations
$V_i S_i$	α	α
$V_i S_i(N)$	βN	βN
$W_i(n)$	$\alpha + \beta$	$2\alpha N + \frac{3\beta}{2}(N^2 + N)$
$P_i(l, n)$	βN	$2\beta(N^2 + N)$
$P_i(0, n)$	β	$\frac{\beta}{2}(N^2 + N)$
$X_0(n)$		$\alpha N + \beta N$
$\bar{\pi}_i(n)$	$\alpha + \beta$	$\alpha N + \beta N$
$X_i(N)$	$\alpha + \beta$	$\alpha + \beta$
$U_i(N)$		$\alpha + \beta$
$R_i(N)$		$\alpha + \beta$
Load indep.	4α	$4\alpha(N + 1)$
Load dep.	$\beta(2N + 4)$	$\beta(4N^2 + 9N + 3)$

Table 4

Modified MVA - Storage and Operations

item	storage	no. operations
$V_i S_i$	α	α
$\bar{\pi}_i(N)$	$\alpha + \beta$	$2\alpha N + \beta(N^2 + N)$
$G(N)$		$\alpha N + \beta N$
$P_i(l, N)$	βN	$\beta(N^2 + N)$
$P_i(0, N)$	β	$\frac{\beta}{2}(N^2 + N)$
$X_i(N)$		$\alpha + \beta$
$U_i(N)$		$\alpha + 2\beta$
$\bar{\pi}_i(N)$		$\alpha + \beta$
$R_i(N)$		$\alpha + \beta$
Load indep.	2α	$\alpha(3N + 5)$
Load dep.	$\beta(N + 2)$	$\frac{\beta}{2}(5N^2 + 7N + 10)$

Table 5

LBANC Method - Storage and Operations

to apply MVA to most forms of product form network; the exceptions are external arrival rates which depend on the population and load dependent servers whose service rate depends on the number of jobs in each class at the device (as opposed to the total number of jobs at the device). Chandy and Sauer [ChSa81] have given the LBANC algorithm for all product form networks except for these two types. Recently, this has been extended to include jobs from open classes at load dependent devices [Saue81].

<u>N</u>	<u>α</u>	<u>β</u>	<u>method</u>	<u>storage</u>	<u>no.operations</u>
7	5	2	conv.	45	519
			MVA	37	760
			MVA mod.	56	684
			LBANC	28	434
10	10	0	conv.	21	540
			MVA	30	620
			MVA mod.	40	440
			LBANC	20	350
10	0	10	conv.	231	2910
			MVA	140	8820
			MVA mod.	240	4930
			LBANC	120	2900
10	5	5	conv.	126	1725
			MVA	85	4410
			MVA mod.	120	2685
			LBANC	60	1625

Table 6

Examples

Another implementation issue is the ease of understanding and the ease of implementation. Here, MVA is the winner, as it is based on some very intuitive arguments [SaCh80]. The convolution method is the most complex. This difference is especially true when programs are extended to include multiple job classes.

A remaining issue is that of verifying programs are producing correct solutions. Related to this is the issue of problem-dependent errors, usually due to errors caused by the arithmetic properties of computers (numerical stability). While the issue of numerical stability has been discussed in several places, the issue of verifying correct results has received almost no attention. Schwetman [Schw80a] has presented a preliminary approach to developing procedures for testing QNM software. This approach is based on solving a succession of models, each with solutions which be verified by alternative methods.

Each of the three algorithms is known to have a potential for two kinds of

numerical difficulties [ChSa80]. Briefly, the convolution algorithm and LBANC may compute normalizing constants $G(n)$ which exceed the range of numbers representable in a computer word. The effects of this problem can be minimized by a judicious choice of the $\{V_i\}$ (which are unique up to a multiplicative constant). Reiser and Sauer [ReSa78] detail a procedure for scaling the V_i , as do Bruell and Balbo [BrBa80].

The second type of difficulty occurs in the calculation of queue length distributions. This is required in both MVA and LBANC, for devices with load dependent servers. In some cases, the calculation of $P_i(0, n-1)$ becomes inaccurate; because $P_i(n, N)$ is computed using $P_i(n-1, N-1)$, these errors propagate, leading to "chaotic behavior" [ChSa80]. Similar problems can occur in the calculation of the auxiliary functions $g_k^i(n)$ in the convolution algorithm for load dependent devices.

Discussion

This paper has presented three solution algorithms for single class, closed queueing network models. The resource usage for each of them was estimated, in terms of the amount of storage and the number of operations. Other issues relevant to implementing each solution were discussed. The conclusion is that there is currently no single algorithm which is "best" in all situations. Fortunately, the difficulties which do arise do so only for extreme values of the input parameters. Models of actual systems are usually well-behaved and be reliably solved by these algorithms.

This paper has avoided discussing models with multiple classes of jobs. As would be expected, these cause a great increase in the complexity of the solution programs, the resources required, and the evaluation of each algorithm. Some of these issues are discussed in [BrBa80] and [Zaho80] and are the subject of current research.

4.4

This paper has also avoided the issue of developing realistic and useful models of actual systems. In addition to requiring reliable solution techniques, this activity requires a detailed knowledge of the system being modeled and of a variety of modeling techniques, to handle critical features of models. These considerations have led to the development of approximate solution techniques, for obtaining solutions to extended forms of QNM's. Many of these are based on the MVA algorithm presented above.

References

- [BaBS77] Balbo, G. and S. Bruell, and H. Schwetman, "Customer classes and closed network models - a solution technique", *Proceedings IFIPS 77*, North Holland Pub., 1977, p. 559.
- [BCMP75] Baskett, F., Chandy, K., Muntz, R. and F. Palacios, "Open, closed, and mixed networks of queues with different classes of customers", *Jour. ACM*, (22,2), April, 1975, p. 248.
- [BrBa80] Bruell, S. and G. Balbo, *Computational Algorithms for Closed Queueing Networks*, North Holland Pub., 1980.
- [Buze73] Buzen, J., "Computational algorithms for closed queueing networks with exponential servers", *Comm. ACM*, (16,9), Sept. 1973, p. 527.
- [Buze78] Buzen, J. et al., "BEST/1 - design of a tool for computer system capacity planning", *Proceedings NCC '78*, AFIPS, 1978, p. 447.
- [BuDe80] Buzen, J. and P. Denning, "Operational treatment of queue distributions and mean value analysis", *Computer Performance*, (1,1), June 1980, p. 6.
- [ChSa80] Chandy, K. and C. Sauer, "Computational algorithms for product form queueing networks", *Comm. ACM*, (23,10), Oct. 1980, p.

55

- [DeBu78] Denning, P. and J. Buzen, "The operational analysis of queueing network models", *Comp. Surveys*, (10,3), Sept. 1978, p. 225.
- [KiSe79] Kienzle, M. and K. Sevcik, "Survey of analytic queueing network models of computer systems", *Proc. Conference on Simulation, Measurement and Modeling of Computer Systems*, ACM/SIGMETRICS, August, 1979, p. 113.
- [ReKo75] Resier, M. and H. Kobayashi, "Queueing networks with multiple closed chains: theory and computational algorithms", *IBM Jour. of R. & D.*, (19,3), May, 1975, p. 283.
- [ReLa80] Resier, M. and S. Lavenberg, "Mean value analysis of closed multichain queueing networks", *Jour. ACM*, (27,2), April 1980, p. 313.
- [ReSa78] Reiser, M. and C. Sauer, "Queueing network models: methods of solution and their program implementation", *Current Trends in Programming Methodology: Software Modeling*, (Vol. III), Prentice-Hall, 1978, p. 115.
- [SaCh81] Sauer, C. and K. Chandy, *Computer System Performance Modeling*, Prentice-Hall, 1981.
- [Saue81] Sauer, C., "Computational Methods for Product Form Queueing Networks", a talk presented at the ORSA-TIMS Special Meeting on Applied Probability - Computer Science, the Interface, January 5-7, 1981, at Florida Atlantic University.
- [Schw80a] Schwetman, H. "Testing network-of-queues software", CSD-TR 330, Dept. Computer Sciences, Purdue Univ., 1980.
- [Schw80b] Schwetman, H., "Implementing the mean value algorithm for the solution of queueing network models", CSD-TR 355, Dept. Computer Sciences, Purdue University, 1980.

[Zaho80] Zahorjan, J., "The approximate solution of large queueing network models", Ph.D. Thesis, Dept. of Computer Science, Univ. of Toronto, 1980; available as Technical Report CSRG-122, Computer Systems Research Group, University of Toronto.